CSC/MAT-220: Lab 3

Due: 9/24/2018

LaTeXis a typesetting program written by Leslie Lamport that is an extension of the original TeXprogram written by Donald Knuth. Hopefully, you have already seen the power that LaTeXoffers by providing powerful macros for typesetting math characters and formulae, and providing packages for the complete control over the formatting of your document. In this lab, we will gain additional experience with LaTeXby working with the PythonTeX and TikZ packages.

PythonTeX

PythonTeX is a TeX package that allows Python code to be run within a document, typesetting the results. This package is currently maintained by Geoffrey Poore and you can visit the github/wiki page.

In order to use PythonTex you must add the following preamble to your tex file.

```
\usepackage{pythontex}
```

In addition, to compile a LaTeX file that uses PythonTeX you will need to access your terminal, change directories to the one with your tex file, and run the following command.

pdflatex file.tex && pythontex file.tex && pdflatex file.tex

Py and Pyc Macros.

The macro \py{expression} evaluates a Python expression and typeets its value. As an example, consider the following snippet of code.

```
1 The Binomial Coefficient ``5 choose 2'' is $\binom{5}{2}=\py{5*4/2}$
```

The Binomial Coefficient "5 choose 2" is $\binom{5}{2} = 10.0$

The macro \pyc{expression} evaluates a Python expression and typesets anything that it prints.

```
1 The Binomial Coefficient ``5 choose 2'' is $\binom{5}{2}=\pyc{import math; x=math.factorial(5); y
      =math.factorial(2)*math.factorial(5-2); print(x/y)}$
```

The Binomial Coefficient "5 choose 2" is $\binom{5}{2} = 10.0$

Note the subtle difference between the macros \py and \pyc. Because the latter only typesets what the Python expression prints, we can do some behind the scenes work and then print only what we want to be typed.

Pycode Macro.

For more lines of Python code, it is ideal to use the \pycode environment. As with the macro \py, only what is printed gets typesetted. Consider the snippet of code below which generates a table with a loop.

```
\begin{pycode}
import math
print(r"\begin{tabular}{c | c | c | c}")
print(r"$n$ & $n!$ & Stirling's Formula & Relative Error \\ \hline")
for n in range(10,51,10):
    x=math.factorial(n)
    y=math.sqrt(2*math.pi*n)*(n/math.e)**(n)
    z=math.fabs((x-y)/y)
    print(r"%d & %e & %e & %e \\\hline" % (n,x,y,z))
print(r"\end{tabular}")
\end{pycode}
```

n	$n!$	Stirling's Formula	Relative Error
10	3.628800e+06	3.598696e+06	8.365359e-03
20	2.432902e+18	2.422787e+18	4.175011e-03
30	2.652529e+32	2.645171e+32	2.781536e-03
40	8.159153e+47	8.142173e+47	2.085461e-03
50	3.041409e+64	3.036345e+64	1.668034e-03

We can also use pycode to create a function that we make use of outside of the pycode environment. In the following example, we define a function in the pycode environment and then use it later with the py macros.

```
\begin{pycode}
def fib(n):
    if n==0 or n==1:
        return 1
    else:
        return fib(n-1)+fib(n-2)
\end{pycode}
Table of the first 5 Fibonacci numbers:
\begin{tabular}{c | c | c | c | c | c}
$n$ & 0 & 1 & 2 & 3 & 4\\
\hline
$F_{n}$ & \py{fib(0)} & \py{fib(1)} & \py{fib(2)} & \py{fib(3)} & \py{fib(4)}\\
\hline
\end{tabular}
```

Table of the first 5 Fibonacci numbers:

n	0	1	2	3	4
F_n	1	1	2	3	5

Sessions. Py, Pyc, and Pycode all have an optional session argument, this is why we are able to define a function inside of pycode and use it outside of pycode, as was done in the previous example. If no session argument is given, then the default session is used. Sessions may be used to speedup compile time, since sessions with different names may be executed in parallel.

```
\begin{pycode}[plot1]
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
ax = plt.axes(projection='3d')
ax = plt.axes(projection='3d')

# Data for a three-dimensional line
zline = np.linspace(0, 15, 1000)
xline = np.sin(zline)
yline = np.cos(zline)
ax.plot3D(xline, yline, zline, 'gray')

# Data for three-dimensional scattered points
zdata = 15 * np.random.random(100)
xdata = np.sin(zdata) + 0.1 * np.random.randn
    (100)
ydata = np.cos(zdata) + 0.1 * np.random.randn
    (100)
ax.scatter3D(xdata, ydata, zdata, c=zdata, cmap
    ='Greens');
plt.savefig('plot1.pdf')
\end{pycode}
\IfFileExists{plot1.pdf}
{
    \includegraphics[scale=0.25]{plot1.pdf}
}
{
    \emph{plot1.pdf not found}
}
```

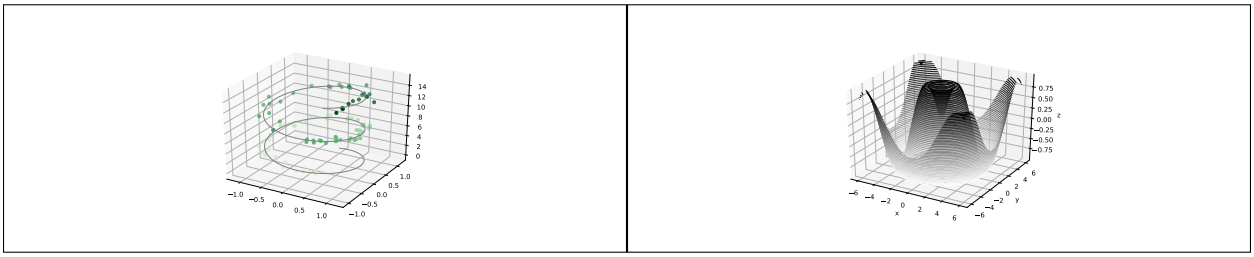
```
\begin{pycode}[plot2]
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt

# Function
def f(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))

# Data for grid
x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)
X, Y = np.meshgrid(x, y)

# Function values on grid
Z = f(X, Y)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.contour3D(X, Y, Z, 50, cmap='binary')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
plt.savefig('plot2.pdf')
\end{pycode}
\IfFileExists{plot2.pdf}
{
    \includegraphics[scale=0.25]{plot2.pdf}
}
{
    \emph{plot2.pdf not found}
}
```



Note the use of the session names: *plot1* and *plot2* in the above example, and the use of the macro `\IfFileExists{}`. Multiple sessions are independent, they do not share variables or function definitions. Sometimes, this is exactly what we want, other times we would like to have global definitions that are available to all sessions. This is what the `\pythontexcustomcode` environment is for. For our last example, we will create a global function for finding all the prime numbers between two integers *start* and *end*. We will first use this function to list all primes between 1 and 100, and then we will use it to create a table listing the number of primes in a given interval.

```
% global function
\begin{pythontexcustomcode}{py}
def FindPrimes(start,end):
    x=[]
    if start==1:
        start=start+1
    for i in range(start,end+1):
        check=True
        for j in range(2,i):
            if i%j==0:
                check=False
                break
        if check==True:
            x=x+[i]
    return x
\end{pythontexcustomcode}
% session prime 1
\begin{pycode}[prime1]
print(r"\begin{tabular}{c | c}")
print(r"Interval & Number of Primes \\ \hline")
for n in range(1,5001,1000):
    x=len(FindPrimes(n,n+999))
    print(r"[%d,%d] & %d \\ \hline" % (n,n+999,x))
print(r"\end{tabular}")
\end{pycode}
% session prime 2
\begin{pycode}[prime2]
print(r"\begin{tabular}{c | c}")
print(r"Interval & Number of Primes \\ \hline")
for n in range(5001,10001,1000):
    x=len(FindPrimes(n,n+999))
    print(r"[%d,%d] & %d \\ \hline" % (n,n+999,x))
print(r"\end{tabular}")
\end{pycode}
% session prime 3
\begin{pycode}[prime3]
print(r"\begin{tabular}{c | c}")
print(r"Interval & Number of Primes \\ \hline")
for n in range(10001,15001,1000):
    x=len(FindPrimes(n,n+999))
    print(r"[%d,%d] & %d \\ \hline" % (n,n+999,x))
print(r"\end{tabular}")
\end{pycode}
```

Interval	Number of Primes	Interval	Number of Primes
[1,1000]	168	[5001,6000]	114
[1001,2000]	135	[6001,7000]	117
[2001,3000]	127	[7001,8000]	107
[3001,4000]	120	[8001,9000]	110
[4001,5000]	119	[9001,10000]	112

Interval	Number of Primes
[10001,11000]	106
[11001,12000]	103
[12001,13000]	109
[13001,14000]	105
[14001,15000]	102

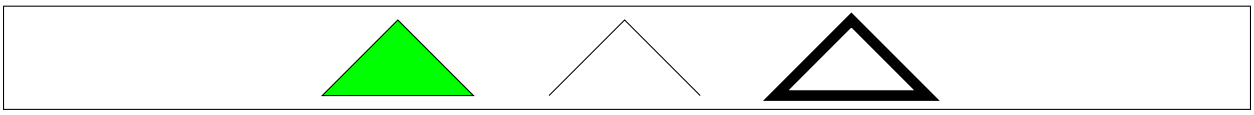
TikZ

TikZ is a recursive acronym for "TikZ ist kein Zeichenprogramm." Now that you know what TikZ is not, know that TikZ is a set of higher level macros that use a lower level language PGF to create vector graphics. The designer of these languages, Professor Till Tantau, is also the creator of the TeX interpreter, which allows us to integrate these graphics within our LaTeX file. You can gain access to the TikZ macros by including the *usepackage{tikz}* line at the top of your LaTeX file. There are two basic elements of TikZ that we will make extensive use of: *paths* and *nodes*.

Paths

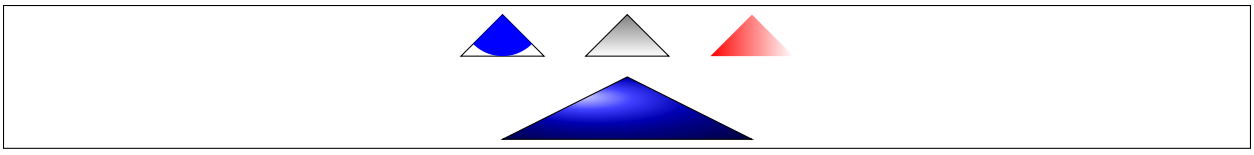
A path is a series of straight and curved line segments, connecting points within a coordinate system. See the example below.

```
\begin{tikzpicture}
        \path[draw] (0,0) -- (1,1) -- (2,0);
        \path[draw,line width=4pt] (3,0) -- (4,1) -- (5,0) --cycle;
        \path[draw, fill=green] (-1,0) -- (-2,1) -- (-3,0) -- cycle;
\end{tikzpicture}
```



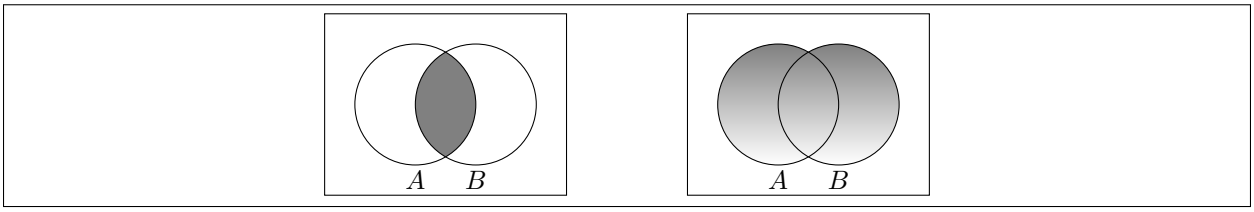
Note how the positioning of each figure is referenced by the origin $(0, 0)$ of the coordinate system, and the use of semicolons to end each command. This picture can be made larger or smaller using the scaling option. In the following example, we highlight the use of several shading and clipping options, while using the scaling option to make the graphics fit within our desired window.

```
\begin{tikzpicture}[scale=0.55]
        %Top Row
        \path[shade,draw] (0,0) -- (1,1) -- (2,0) -- cycle;
        \shade[left color=red] (3,0) -- (4,1) -- (5,0) -- cycle;
        \begin{scope}
                \path[clip, draw] (-1,0) -- (-2,1) -- (-3,0) -- cycle;
                \path[fill=blue] (-2,1) circle (1);
        \end{scope}
        %Bottom Row
        \path[draw, shading=ball, ball color=blue] (-2,-2) -- (1,-0.5) -- (4,-2) -- cycle;
\end{tikzpicture}
```



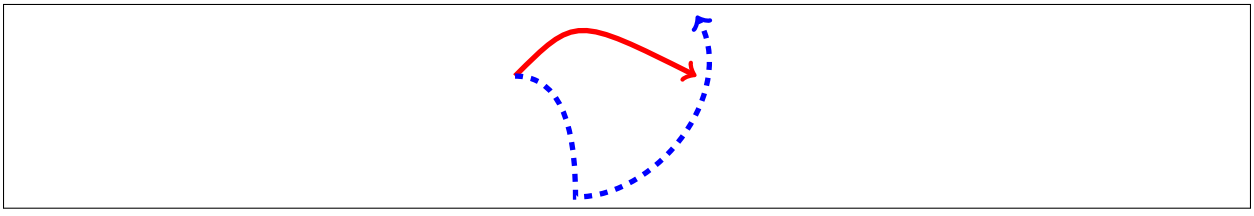
Note the scope control on the clipping in the top row, this is done so that the clipping does not effect the bottom row command. Second, note the shading within the path on the bottom row, in contrast to the shade in the top row, which does not outline a path. Lastly, note the circle command places a circle centered at (-2,1) with radius (1) within the predefined path clipping. To show the power in clipping and scope, we use these commands along with some basic shapes to create a Venn Diagram of the intersection of two sets.

```
\begin{tikzpicture}[scale=0.8]
        %Intersection
        \draw (-5,-1.5) rectangle (-1,1.5);
        \begin{scope}
                \clip ( -3.5,0) circle (1);
                \clip (-2.5,0) circle (1);
                \fill[color=gray] (-5,-1.5) rectangle (-1,1.5);
        \end{scope}
        \draw (-3.5,0) circle (1);
        \draw (-2.5,0) circle (1);
        \node at (-3.5,-1.25){$A$};
        \node at (-2.5,-1.25){$B$};
        % Union
        \draw(1,-1.5) rectangle (5,1.5);
        \shade[color=gray] (2.5,0) circle(1);
        \shade[color=gray] (3.5,0) circle(1);
        \draw (2.5,0) circle(1);
        \draw (3.5,0) circle(1);
        \node at (2.5,-1.25){$A$};
        \node at (3.5,-1.25){$B$};
\end{tikzpicture}
```



Note the use of nodes, which we will discuss in the next section. Lastly, we can also do curved paths connecting points by specifying control points, or specifying directions in and out of points. The in and out parameters are given in terms of angles relative to the start and end points, we specify each path by its color and style.

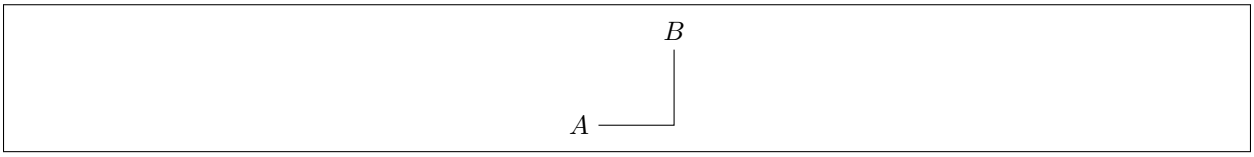
```
\begin{tikzpicture}[scale=0.8]
        \draw[line width=2pt, color=red, ->] (0,0) .. controls(1,1) .. (3,0);
        \draw[line width=2pt, color=blue, dashed, ->] (0,0) to [in=90, out=0] (1,-2) to [in=300,
            out=0] (3,1);
\end{tikzpicture}
```



Nodes

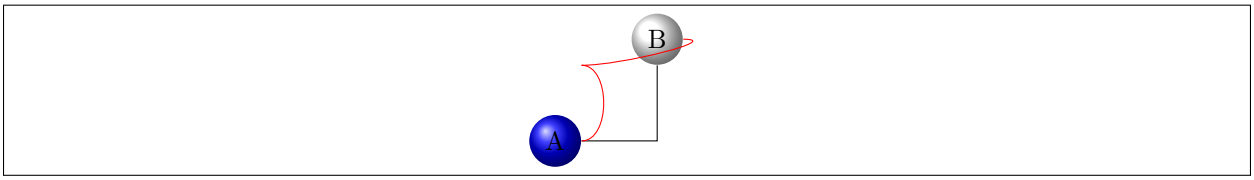
As was done with the Venn Diagrams above, nodes allow us to label our diagrams. Consider the example below, and note the use of the left and above options, which force the node labels to not overlap with the path.

```
\begin{tikzpicture}
        \path[draw] (0,0) node[left] {$A$} -- (1,0) -- (1,1) node[above] {$B$};
\end{tikzpicture}
```



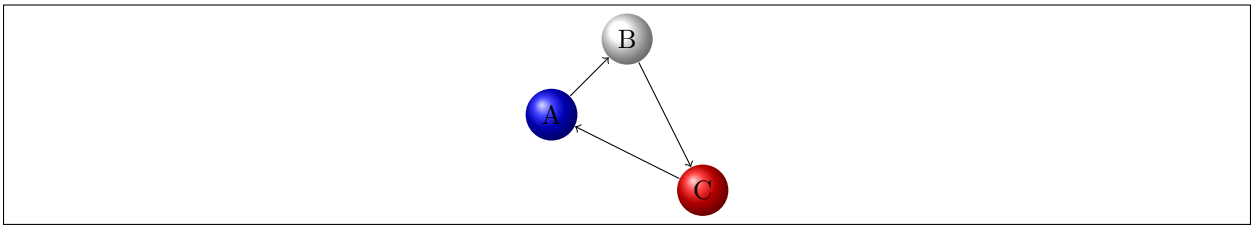
From an organizational standpoint, it is often better to define the nodes first and then use them later.

```
\begin{tikzpicture}
        \node[left, circle, shading=ball, ball color=blue] (a) at (0,0) {A};
        \node[above,circle, shading=ball, ball color=white] (b) at (1,1) {B};
        \path[draw] (a) -- (1,0) -- (b);
        \path[draw,color=red] (a) to [in=0, out=0] (0,1) to [in=0, out=0] (b);
\end{tikzpicture}
```



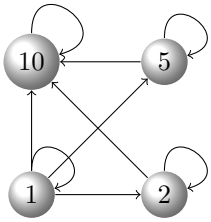
Note that in the previous example, we were able to use the same nodes on multiple occasions; therefore, defining the nodes first also has efficiency benefits. This can be done in a powerful iterative form, consider the following example.

```
\begin{tikzpicture}
        \node[circle, shading=ball, ball color=blue] (a) at (0,0) {A};
        \node[circle, shading=ball, ball color=white] (b) at (1,1) {B};
        \node[circle, shading=ball, ball color=red] (c) at (2,-1) {C};
        % Graph
        \foreach \from/\to in {a/b, b/c, c/a}
                \draw[->] (\from) -- (\to);
\end{tikzpicture}
```



There is so much more to learn about TikZ, but for now you have the basics necessary to create Venn diagrams, relational graphs, paths, and other shapes. There is so much information online it can be overwhelming. A good source is the ShareLaTeX page and TikZ manual. In conclusion, we draw the relation graph from [1, 14.17 a]. Note the use of *tikzstyle* and *looseness*, and also the coordinate scale setting of x=5em, y=5em.

```
\begin{tikzpicture}[x=5em,y=5em]
	%set style options the same for each node
	\tikzstyle{every node}=[circle, shading=ball, ball color=white];
	\node (a) at (0,0) {$1$};
	\node (b) at (1,0) {$2$};
	\node (c) at (1,1) {$5$};
	\node (d) at (0,1) {$10$};
	%draw (a,b), (a,c), (a,d), (b,d), and (c,d)
	\foreach \from/\to in {a/b, a/c, a/d, b/d, c/d}
		\draw[->] (\from) -- (\to);
	%draw (a,a), (b,b), (c,c), and (d,d)
	\foreach \from/\to in {a/a, b/b, c/c, d/d}
		\draw[->] (\from) to [in=15, out=90, looseness=5] (\to);
\end{tikzpicture}
```



Assignment

Write LaTeX code for each of the following problems. You will be graded on whether or not your code compiles to produce the intended PDF file, and on the organization and clarity of your code. The file you turn in should be labeled as follows: *yourname_labnumber.tex*.

I. Use the `\pythontexcustomcode` to write a function titled Pascal that is available to all PythonTex sessions. The function should return the $n \geq 0$ row of Pascal's triangle as a list, where n is the parameter of the function. Then, use this function in two parallel sessions: The first session should call the function to print out rows $n = 0, 1, \ldots, 5$ of Pascal's triangle. The second session should use the function to print the result of $\binom{100}{7}$.

II. Use TikZ to draw a Venn diagram for the set operations $A \triangle B$ and $A \cap B \cap C$.

III. Use TikZ to draw a relational graph for [1, 14.17 b].

References

[1] E. R. Scheinerman, *Mathematics: A discrete introduction*, 3rd ed., Brooks/Cole, Boston, MA, 2013.